Security Testing of a Secure Cache Design

Fangfei Liu
Princeton University
Princeton, NJ, 08544 USA
fangfeil@princeton.edu

Ruby B. Lee
Princeton University
Princeton, NJ, 08544 USA
rblee@princeton.edu

ABSTRACT

Cache side channel attacks are attacks that leak secret information through physical implementation of cryptographic operations, nullifying cryptographic protection. Recently, these attacks have received great interest. Previous research found that software countermeasures alone are not enough to defend against cache side channel attacks. Secure cache designs can thwart the root causes of cache side channels and are more efficient. For instance, Newcache is a cache design that can enhance security, performance and power efficiency simultaneously through dynamic memory-cache remapping and eviction randomization. However, these cache designs seldom had their security verified experimentally by mounting cache side channel attacks on them.

In this paper, we test the security of Newcache using representative classes of cache side channel attacks proposed for conventional set-associative caches. The results show that Newcache can defeat all these attacks. However, what if a very knowledgeable attacker crafted the attack strategy targeting the secure cache's design? We redesign the attacks specifically for Newcache. The results show that Newcache can defeat even crafted access-driven attacks specifically targeted at it but sometimes succumbs to the specifically crafted timing attacks, which is due to a very subtle vulnerability in its replacement algorithm. We further secure Newcache by modifying its replacement algorithm slightly, thus defeating these specifically crafted timing attacks. In addition, the improved Newcache simplifies the replacement algorithm in the original Newcache design.

1. INTRODUCTION

Encryption is used to protect the confidentiality of secret information. However, the protection of strong cryptography may be nullified by the physical implementation of the cipher. The byproduct information of cryptographic operation such as timing, power, and eletromagnetic radiation can be exploited by attackers to extract secret information. These are called side channel attacks[7, 4]. Cache side channel attacks are side channel attacks based on cache access mechanisms in processors[13, 15, 2, 12, 3, 6, 18]. Many cryptographic operations heavily use memory accesses to look up substitution tables, and the addresses of the memory accesses are dependent on the secret key. Therefore, if the address of the memory accesses are known to the attackers, it is possible to infer secret key bits. The use of a cache enables attackers to learn the addresses of memory accesses since the timing difference between cache hits and cache misses is large. Cache side channel attacks can be performed on various platforms, from smart cards to cloud server, and therefore are a serious security consideration.

Most existing solutions for cache side channel attacks are software based. Some advancements have been made for high performance AES implementations without table lookups. For example, Könighofer et al proposed a fast implementation of AES using bitslices with constant time [9]. Intel introduced AES-NI instructions that enable fast software AES implementation using AES-specific instructions [11]. However, these solutions are specific to AES. Lee [10] proposed a general-purpose *parallel table lookup* instruction, that allows software-controlled tables to be accessed in constant time, speeding up AES and avoiding cache side channel attacks. Recently, several hardware secure cache designs have been proposed to secure the cache itself. The hardware solutions are more general and apply to different cryptographic operations and attacks without causing significant performance degradation. Although security analysis specific to some secure cache designs is provided in some previous work[17, 16, 5, 8, 13], few designs have been tested with real attack experiments. Mounting practical cache side channel attacks on a secure cache design is the most straightforward way to evaluate its security, albeit it cannot formally prove a design is secure if the attack does not succeed, given a fixed number of trials that an attacker can perform. However, it does identify the potential security vulnerability of the design if the attack does succeed. Therefore, it is very helpful in guiding secure cache design for computer architects. It is worth noting that it is non-trival to mount the proposed cache side channel attacks. Nearly all the proposed cache side channel attacks are based on the assumption of a conventional set-associative cache, which is no longer true in many secure cache designs. In this paper, we evaluate the security of a secure cache design, Newcache[16], using existing attacks for conventional caches as well as specifically redesigned attacks for Newcache.

The contributions of this work are:

- Evaluating the security of Newcache using practical

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
HASP '13, June 2013, Tel-Aviv Israel
Copyright 2013 ACM 978-1-4503-2118-1/13/06 ...$15.00.

cache side channel attacks for conventional caches and finding that Newcache can defeat all three representative attacks.

- Redesigning attacks specifically for Newcache and finding that Newcache can defend against access-driven attacks but does not always defend against a timing attack crafted specifically for it.
- Improving the random replacement algorithm of Newcache so that it can defeat the specifically designed attacks while using a simpler replacement algorithm than the original Newcache design.

The rest of the paper is organized as follows: in section 2, we briefly introduce representative cache side channel attacks, and the cache architecture of Newcache. In section 3, we show the results for evaluating the security of Newcache using existing attacks. In section 4, we redesign attacks specifically targeting Newcache with the knowledge of Newcache organization. We show the improved Newcache design in section 5 that secures Newcache against the specifically designed attacks. We conclude our work in section 6.

2. BACKGROUND

2.1 Classification of Cache Side Channel Attacks

Cache side channel attacks are usually categorized into three types [1]: *access-driven*, *trace-driven*, and *timing-driven* attacks. In access-driven attacks, the adversary can get information on which cache sets are accessed by the victim process using a technique called the *Prime and Probe* attack [12] shown in Figure 1a. The adversary runs a spy process on the same processor as the victim process. The spy process repeatedly "primes" the cache by filling the cache with its own data. After a certain time interval, the spy process runs again, and measures the time to load each set of its data (called the "Probe" phase, which primes the cache for subsequent observations.). If the victim process accesses some set of the cache during this time interval between Prime and Probe, it will evict at least one of the spy process' cache lines, which causes the spy process to have a cache miss on this cache line, and thus a higher load time for this block.

On the other hand, for trace-driven and timing-driven attacks, an adversary is able to get information related to the total number of cache hits or misses for the victim process' memory accesses. In trace-driven attacks, an adversary can capture the profile of cache activity in terms of hit and miss for each memory access of the victim process performing an encryption. In timing-driven attacks, the adversary can only observe the aggregate profile, i.e, the total execution time for encrypting a block. This approximates the total number of cache hits and misses.

Even for timing-driven attacks, an adversary can introduce some interference to the victim process to indirectly learn whether a certain set is accessed by the victim process. This technique is called *Evict and Time*[12] as shown in Figure 1b. The adversary can first trigger a block encryption and then evict one specific cache set with its own data items. The adversary then triggers another block encryption, and measures the encryption time of the victim process. If the victim process accesses the evicted set, the block encryption time tends to be higher due to incurring

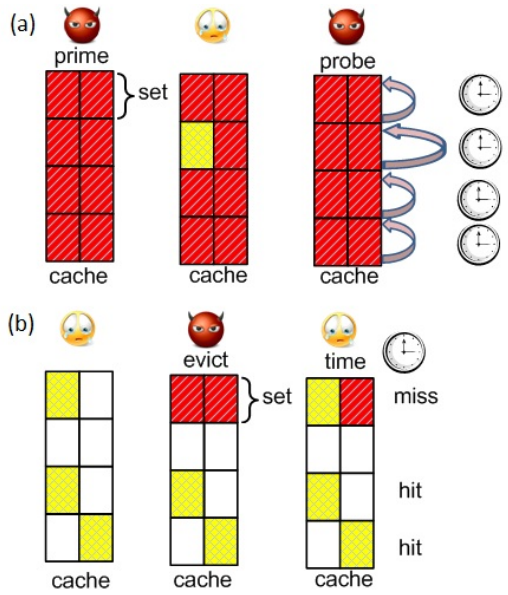

Figure 1: Illustration of (a) Prime and Probe and (b) Evict and Time attack technique

at least one cache miss. Bernstein's attack [2] is a special case of the Evict and Time attack in which the eviction of a cache set is done by the program wrapper which is within the victim process.

A cache collision timing attack is a timing-driven attack where there is no interference by an adversary's process. Without running a spy process, the adversary has to rely on the victim process' cache hits to learn something about the memory addresses used by the victim.

In this work, we will focus on access-driven attacks and Evict and Time attacks because defending against these classes of attacks is the goal of Newcache.

2.2 Newcache Architecture

Several hardware solutions have been proposed to defend against cache side channel attacks[14, 17, 16, 8, 5]. Most of these hardware solutions use some form of cache partitioning or randomization to eliminate the cache interferences or randomize the cache interferences, respectively. In the cache partitioning approach, the cache is partitioned into different zones for different processes and each process can only access its reserved cache lines, thus eliminating cache interference[17, 5]. Instead of not allowing cache interference, the randomization approach randomizes cache interference by random permutation of the memory-cache mapping and randomizing the cache eviction[17, 16]. This is like performing a "Moving Target Defense (MTD)" in hardware cache design, against access-based attacks and eviction-based timing attacks.

Newcache[16] is a representative cache design using the randomization approach, employing both techniques of dynamic memory-cache remapping and eviction randomization. The block diagram and cache access handling process of Newcache is shown in Figure 2a and Figure 3a. Newcache introduces a level of indirection in the memory-cache mapping by employing a logical direct mapped (LDM) cache (which does not exist physically). The LDM cache has a

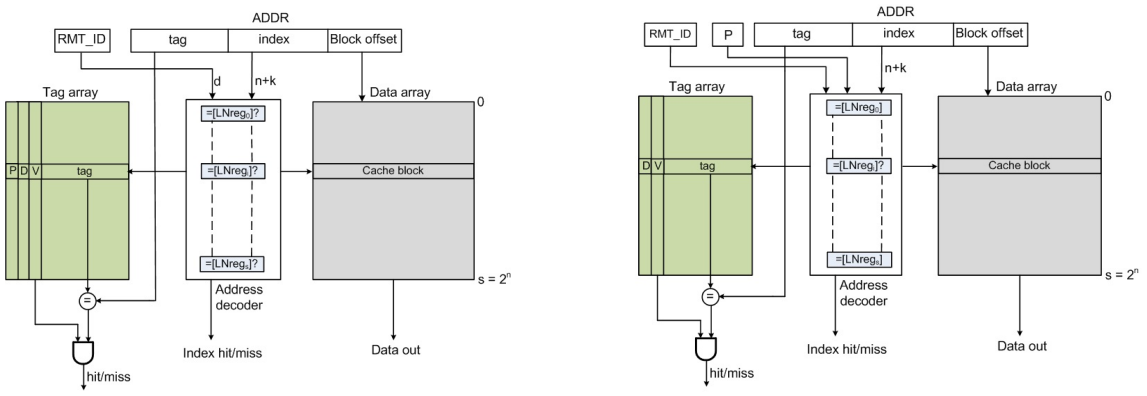

Figure 2: Block diagrams of (a) original Newcache and (b) improved Newcache

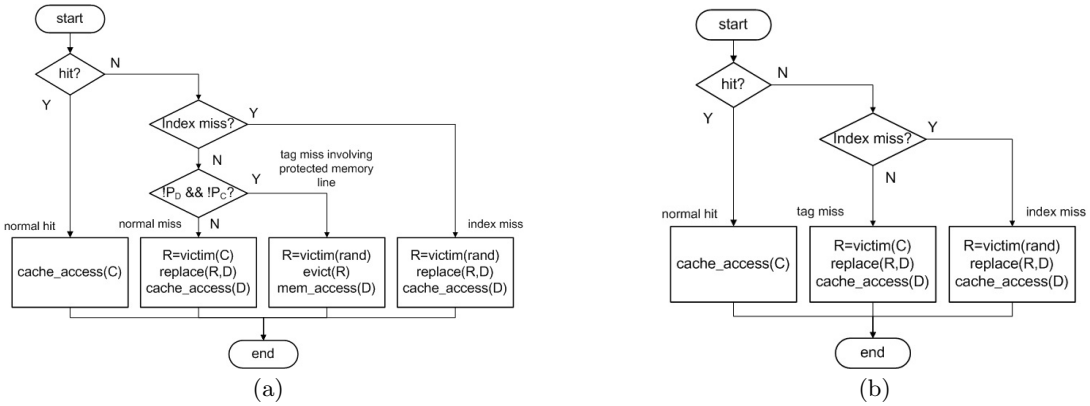

Figure 3: Cache access handling process of (a) original Newcache and (b) improved Newcache. C: cache line selected by address decoder; D: memory line accessed; R: victim cache line for replacement; P_X: protection bit of line X

larger size than the physical cache and is indexed using a longer index than needed for the physical cache. Hence the memory-cache mapping consists of a fixed mapping from the memory to the ephemeral cache and a fully-associative mapping from the LDM to the physical cache. To implement Newcache, the address decoder in a conventional set-associative cache is replaced by a remapping table that contains a set of line number registers(LNregs). Each LNreg stores a unique mapping from a logical direct mapped (LDM) cache line to the physical cache line. The same set of LNregs can be shared by multiple remapping tables (identified by RMT_ID), so that an attacker observes a different memory-cache mapping than the victim. Each cache access starts at searching the contents of the LNregs using the index bits and the RMT_ID. If no matched LNreg is found, an index miss occurs. To randomize the cache interference, a random victim cache line is selected for replacement, which at the same time will randomly permute the memory-cache mapping. If a mapping is found in the LNreg, the following access is similar to accessing a direct mapped cache, which may have a tag miss. With the use of multiple remapping tables and eviction randomization, external cache interferences are totally randomized. However, internal cache interferences may still exist through tag misses. To solve this issue, Newcache associates a protection bit in the tag array for each cache line to indicate whether this cache line contains security-critical data. If a tag miss involves protected data, the missing cache line will not be brought into the cache and will be forwarded directly to the processor. Instead, a random cache line is evicted, tricking an attacker into thinking that the missing memory line replaced a random line.

3. ANALYZING ATTACKS

A complete cache side channel timing attack consists of an online phase and an offline phase. In the online phase, the attacker collects measurement samples during the victim's encryption. In the offline phase, the attacker processes the measurement samples to recover secret keys. Only the online phase depends on the cache organization and the offline phase is uniquely determined by the attack strategy. Therefore, we focus on the online phase of the *Evict and Time* and *Prime and Probe* attacks and detail its strategy for getting measurements.

3.1 General Measurement Strategy

3.1.1 Evict and Time attacks

The purpose of the *Evict and Time* attack is to learn whether a specific security-critical memory line E is accessed during encryption, therefore it is important for the attacker to access appropriate memory lines in his memory space before each encryption, which can occupy all the possible cache slots where E may be placed, so that:

- memory line E is guaranteed to not be present in the cache before encryption
- other security-critical memory lines (denoted as O) may remain in the cache before encryption.

Then if memory line E is accessed in the subsequent encryption (depending on the secret key), it will lead to cache miss when the attacker accesses it again,, thus it will have a higher encryption time. For a conventional set-associative (SA) cache, a cache set contains all the possible cache slots that a specific memory line can be placed. Assume the cache line size is B bytes, there are S cache sets in total and each cache set contains W cache lines, then any two memory lines that are $S \cdot B$ byes apart will be mapped to the same cache set. If the attacker allocates an array of the cache size and accesses W such memory lines, the specific cache set will be exclusively occupied by the attacker's memory lines.

3.1.2 Prime and Probe attacks

There are two types of information that an attacker can learn from *Prime and Probe* attacks:

(type-I) whether a specific security-critical memory line E is accessed during encryption

(type-II) which security-critical memory lines are accessed during the encryption.

A type-I attack is very similar to the *Evict and Time* attack:

- the attacker needs to prime all the possible cache slots where E can be placed using a specific set of W memory lines in his memory space during the prime stage
- the W memory lines are accessed in a proper order during the probe stage to minimize cache interference within the W memory lines.

Then if the memory line E is accessed during the encryption (depending on the secret key), it will always evict one of the primed memory lines, leading to a higher probe time.

For type-II attacks, assume that for any security-critical memory line E_i $(i = 1, ..., N)$, denote the set of attacker's memory lines that can occupy all the possible cache slots where E_i may be placed as S_i, then we have:

- the set of memory lines (Z) that is accessed during the prime stage satisfies $\bigcup_{i=1}^{N} S_i \subseteq Z$
- the size of Z should be equal or less than the size of the cache to minimize cache interference within Z
- each S_i is probed separately during the probe stage.

If there is no cache interference between S_i and S_j, a type-II attack is exactly a combination of several independent type-I attacks, which is the case for a set-associative (SA) cache. For a SA cache, the W cache slots in a cache set set_i is where E_i can be placed, thus S_i contains W memory lines that are mapped to the same cache set set_i. The union of S_i is a subset of an array of the same size as the cache, which illustrates why the attacker can prime the whole cache with an array of the cache size and probe the cache after the victim program has run, to find the time to load each set, in the *Prime and Probe* attack against a SA cache.

Table 1: Simulator Configurations

Parameter	Value
ISA	x86
Baseline cache size	32 KB
Baseline cache line size	32 B
# of MSHR entry / # of targets per MSHR	1/1
Cache hit latency	1 ns
Memory access latency	100 ns
Number of extra index bits (k) in LNregs	4
Number of measurement samples	2^{20}

3.2 Experimental Results

We now apply the above attacks for a SA cache to Newcache. In particular, we perform simple first-round synchronous attacks against AES-128. The attacker can operate synchronously with the encryption on the same processor and exploit the fact that in the first round of AES encryption, the accessed table indicies $x_i^{(0)}$ are related with plaintext bytes p_i and key bytes k_i as $x_i^{(0)} = p_i \oplus k_i$ for all $i = 0, ..., 15$. Thus, if the plaintext byte p_i and the accessed table index $x_i^{(0)}$ are known to the attacker, the attacker can immediately get the key byte k_i. The plaintext can be recorded by the attacker and the information on the accessed table index is obtained through his measurement of the access time of different cache lines.

We implemented Newcache on gem5, which is a cycle-accurate architectural simulator. We use the classic memory system in gem5 to model Newcache as a non-blocking cache. We used a system shown in Table 1. To increase the latency difference of cache hit and cache miss, we only use a 1-level cache hierarchy and assume 1-ns L1 cache access latency and 100-ns memory access latency. To minimize the latency-hiding effect due to a non-blocking cache, the miss queue only has one miss status handling register (MSHR) and each MSHR can only hold one request. We extended the page table and TLB with one bit to indicate whether a page is protected or not, and the protection bit can be read out from the TLB. We set all the key bytes $k_i = 0$ for all $i = 0, ..., 15$ in the experiment, which does not lose generality since the key byte will be XORed with the random plaintext byte as the table lookup indices for the first round encryption.

3.2.1 Evict and Time attack

For each measurement sample, the attacker needs to record the encryption time of a block encryption and the corresponding 16 plaintext bytes. Then for each plaintext byte p_i, $i = 0, ..., 15$, we average the encryption time for each of the 256 possible values of p_i. In the first round of AES encryption, four table indices $x_l^{(0)}, x_{l+4}^{(0)}, x_{l+8}^{(0)}, x_{l+12}^{(0)}$ will be used to index table T_l, for $l = 0, 1, 2, 3$. Therefore, if the

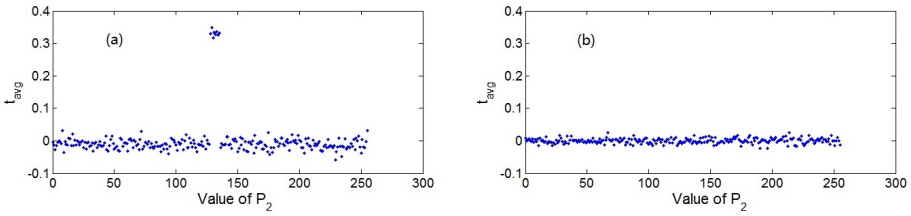

Figure 4: Original *Evict and Time* attack result for (a) conventional 8-way set-associative cache (b) original Newcache design

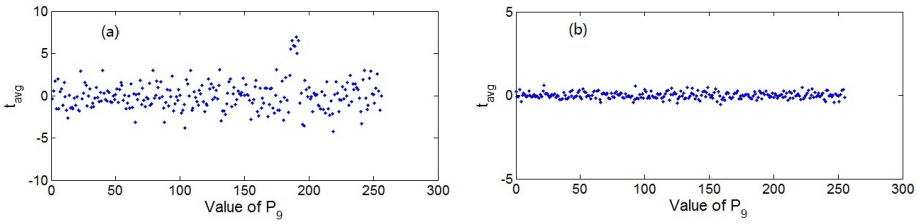

Figure 5: Original Type-I *Prime and Probe* attack result for (a) conventional 8-way set-associative cache (b) original Newcache design

evicted memory line contains table entries of T_l, four table lookups $x_l^{(0)}, x_{l+4}^{(0)}, x_{l+8}^{(0)}, x_{l+12}^{(0)}$ will be impacted, hence there will be a significantly higher average encryption time for certain values of plaintext bytes $p_l^{(0)}, p_{l+4}^{(0)}, p_{l+8}^{(0)}, p_{l+12}^{(0)}$ if the attack can succeed.

Figure 4a and Figure 4b show the results for a conventional 8-way set-associative (SA) cache and Newcache, respectively. For the 8-way SA cache, the encryption time is significantly higher than the average encryption time (only the result for P_2 is shown) when the plaintext takes values from 128 to 135 (8 table entries are indistinguishable since they are in the same cache line), which means that the attacker can easily recover the higher 5 bits of the corresponding key byte. However, for Newcache, no points with significantly higher encryption time can be found for all the plaintext bytes, which means that the (non-crafted) *Evict and Time* attack for the conventional SA cache fails on Newcache. The failure of the attack is due to the fact that the dynamic remapping in the Newcache will not ensure that the W memory lines accessed by the attacker always occupy all the possible cache slots where the security-critical memory line E may be placed.

3.2.2 Prime and Probe attack

We performed attack experiments for both type-I and type-II *Prime and Probe* attacks. For each measurement sample, the attacker needs to record the time to probe each set S_i and the corresponding plaintext bytes. The offline analysis of the type-I attack is the same as the *Evict and Time* attack. As shown in Figure 5a and Figure 5b, the type-I attack succeeded on the 8-way SA cache but failed on Newcache.

The offline analysis of a type-II attack is slightly different. For each plaintext byte p_i, $i = 0, ..., 15$, we average the probe time for a given value of the plaintext byte and a given set S_i and subtract the average time for all values of the plaintext byte, and then use a grayscale figure to represent the probe time (the longer the probe time, the lighter the color). Since we use all-zero key bytes in the experiment, a bright straight line is expected in the grayscale graph if the attack can succeed. For each non-zero key byte, there will be a specific pattern in the grayscale figure, which makes it easier for the attacker to figure out the secret key. The results are shown in Figure 6a and Figure 6b and again the type-II attack succeeded on the 8-way SA cache but failed on Newcache. The failure of the *Prime and Probe* attack is due to the fact that the random eviction in Newcache will not ensure that one of the W memory lines in each set S_i probed by the attacker is always evicted by the specific security-critical memory line E, accessed by the victim.

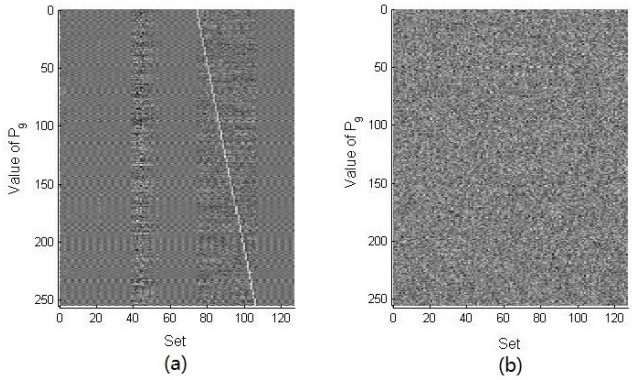

Figure 6: Original Type-II *Prime and Probe* attack result for (a) conventional 8-way set-associative cache (b) original Newcache

4. REDESIGNING ATTACKS TARGETING NEWCACHE

Although all the three attacks for the SA cache fail on Newcache, it does not mean that Newcache can defeat all possible attacks. A smarter attacker may design attacks

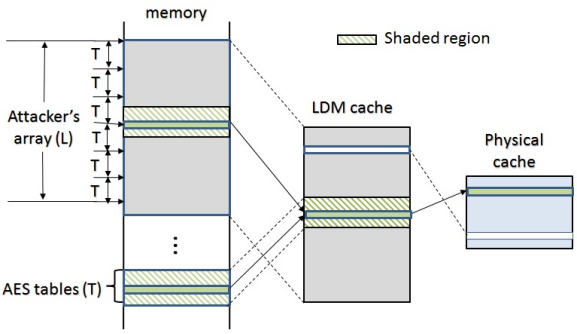

Figure 7: Illustration of *Evict and Time* attack specifically targeting Newcache

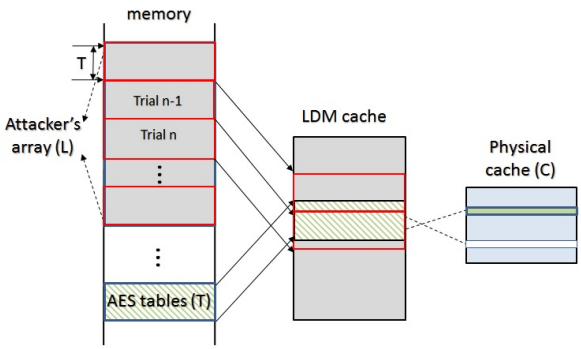

Figure 8: Illustration of *Prime and Probe* attack specifically targeting Newcache, showing that the conflicting memory addresses to be primed by the attacker can come from two trials,$n-1$ and n, each of size T.

specifically for Newcache with the knowledge of its cache organization. From section 3.1 we find that an appropriate measurement for the *Evict and Time* and *Prime and Probe* attacks requires the attacker to evict and probe appropriate sets of memory lines. The W memory lines for the SA cache may not be the right ones for Newcache. We noticed that although the mapping from the LDM to the physical cache is fully associative, the mapping from the memory to the LDM is fixed; Hence a memory line only has one fixed cache slot to be placed in if the mapping from the LDM cache to the physical cache has been established (i.e. index hit but tag miss). This enables an attacker to find the appropriate memory lines by manipulating an array of the same size as the ephemeral LDM cache.

4.1 Evict and Time attacks

To occupy the cache slot that may be replaced by a security-critical memory line E, the attacker only needs to access a memory line which has the same mapping (index bits) as E, which exists in an array of the same size as the LDM cache. As shown in Figure 7, assume that the size of the LDM cache is L bytes and the AES tables are T bytes in total. The diagonally shaded region in Figure 7 represents the memory lines that have the same mapping with certain memory lines in AES tables. Therefore, the attacker can access any one of the memory lines in the shaded region. However, since the attacker does not know the start address of the AES tables and the size of the AES tables T is usually much smaller than the size of the LDM cache L, the attacker has to perform L/T independent trials to locate the desired memory line. A smarter attacker can perform W trials at the same time since at most one of the W trials can evict a cache line containing AES table entries. This attack is summarized as follows:

For the n^{th} trial ($n = 0, 1, ..., L/T/W$), repeat the following steps to collect M measurement samples:

(a) (*Evict*) access some W memory addresses, starting at address $n \cdot W \cdot T$, with T bytes apart.

(b) (*Time*) Trigger an encryption with random plaintext p, and time it.

4.2 Prime and Probe attacks

A type-I *Prime and Probe* attack can be done in a similar way as an *Evict and Time* attack except that the attacker measures the time to probe his own memory line instead of measuring the encryption time. For a type-II attack, each set S_i contains one memory line that has the same mapping as a security-critical memory line E_i. The union of S_i is exactly a continuous array of the same size as the AES tables (T bytes). Since the random eviction in Newcache may lead to cache interference among the probed memory lines, the proper set of memory lines to prime (Z) is the union of S_i (usually smaller than the cache size C). Similar to an *Evict and Time* attack, the attacker does not know the start address of Z and has to do multiple independent trials with the array of the same size as the LDM cache (L bytes) to locate Z. The type-II *Prime and Probe* attack against Newcache is summarized as follows (as shown in Figure 8):

For the n^{th} trial ($n = 0, 1, ..., L/T$), repeat the following steps to collect M measurement samples:

(a) (*Prime*) Read a value in each memory line of the array ranging from $n \cdot T$ to $(n+1) \cdot T$

(b) Trigger an encryption with random plaintext p

(c) (*Probe*) For each set S_i ($i = 0, 1, ..., T/B$), measure the time to access address $(n \cdot T + i \cdot B)$.

Figure 8 also shows that for Newcache, the memory addresses which conflict with AES table entries in the LDM cache, which should be primed by the attacker, can come from two trials, $n-1$ and n, each of size T. The arrows mapping from the LDM cache to the real physical cache is supposed to show that it is random and each physical cache slot is equally likely.

4.3 Experimental Results

4.3.1 Evict and Time attack

Since the size of the LDM cache $L = 512KB$, the AES table size is $T = 4KB$, and we group $W = 8$ trials together, thus we need to perform $L/T/W = 16$ independent trials. In the experiment, the attack succeeded in the 6^{th} trial, the results are shown in Figure 9a. The high T_{avg} numbers near $P_1 = 0$ indicate that Newcache cannot defend against this specifically designed *Evict and Time* attack.

The reason why the attack can succeed for Newcache is very subtle. Assume that the attacker's memory line A has

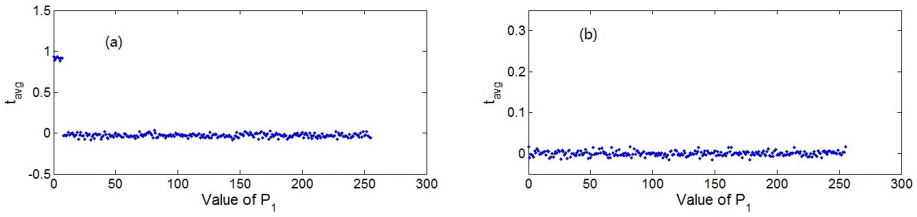

Figure 9: Specifically designed *Evict and Time* attack result for (a) original Newcache design (b) improved Newcache

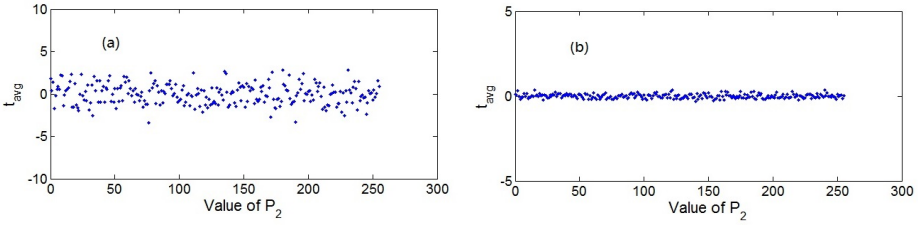

Figure 10: Specifically designed type-I *Prime and Probe* attack result for (a) original Newcache design (b) improved Newcache

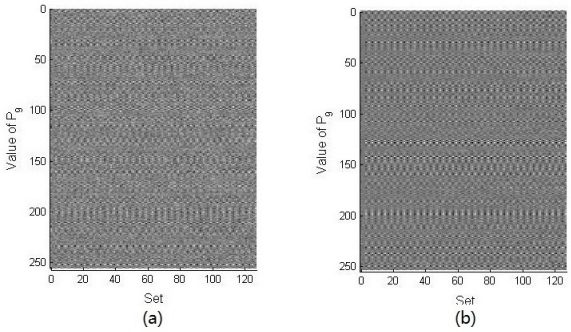

Figure 11: Specifically designed type-II *Prime and Probe* attack result for (a) original Newcache design (b) improved Newcache

the same mapping as memory line E that contains AES table entries. Consider the following scenario: The attacker performs m consecutive measurements and for each measurement the attacker accesses A and then triggers a block encryption; In the $(m+1)^{th}$ measurement, the attacker accesses A again, there are three possible cases for this access:

1. hit if A remains in the cache since last access
2. index miss if both A and E are not in the cache, A will be brought into the cache, randomly replacing a cache line
3. tag miss if E is in the cache, A will not be brought into the cache and a random cache line will be evicted

For the first two cases, all the following accesses to E (not only the first access) during the $(m+1)^{th}$ encryption will have tag miss since the SecRAND algorithm does not allow E to be brought into the cache in this case. Note that the first two cases are highly likely to occur because the SecRAND algorithm cannot prevent the attacker's memory line A from occupying the conflicting cache slot first when neither A nor E is in the cache. The root cause for the success of the attack is that the SecRAND algorithm cannot totally randomize memory-cache mapping.

4.3.2 *Prime and Probe attack*

For the type-I attack, we probe the total time to load $W = 8$ consecutive memory lines with $T = 4KB$ apart in order to reduce the number of trials needed, which leads to $L/T/W = 16$ trials in total. We find that the 3^{rd} trial contains the appropriate memory lines that may have the same mapping as a memory line containing table entries of T_2, and the corresponding plaintext bytes p_2, p_6, p_{10}, p_{14} will potentially be impacted. The result for plaintext byte p_2 is shown in Figure 10a. No significant higher probe time can be found for any plaintext values, which means that even the specifically designed type-I attack cannot succeed on Newcache.

For a type-II attack, we have to perform $L/T = 128$ trials to ensure that the size of the primed array is smaller than the cache size in order to minimize the internal interference of primed memory lines. We find that trial 35 and 36 are the correct trials that contain memory lines which are conflicting with part of the AES tables and we show the result for trial 35 in Figure 11a, where no brighter straight line can be found. Therefore, we can conclude that Newcache significantly increases the difficulty to mount access-driven attacks using *Prime and Probe* techniques.

5. IMPROVED REPLACEMENT ALGORITHM

We propose a minor change to the original Newcache design in [16] to overcome this issue.The security-critical memory lines are distinguished from the non security-critical data within a process by the protection bit (P-bit). Rather than putting the protection bit within the tag array, it is placed in the LNregs, which will be searched before tag comparison, causing an index miss (and subsequent randomization) if the P-bits differ. The block diagram and cache access han-

dling process of this slightly modified Newcache are shown in Figure 2b and Figure 3b, respectively. The P-bit is concatenated with the RMT_ID to determine a remapping table. Note that the index miss check in Figure 3b includes the P-bit equality check. In the above mentioned example, the P-bit of the cache lines containing the AES tables is '1' while it is '0' for all the other data. Therefore, the attacker's memory line A will view a different remapping table from E, thus the internal cache interference through the tag miss is eliminated. Also, the performance is expected to be slightly better than the original Newcache design since it eliminates some of the tag misses (conflicting misses) and extra random evictions. We performed the same attacks against the improved Newcache and show the results in Figure 9b, Figure 10b and Figure 11b. In all the three attacks, no significantly higher encryption time or probe time can be found for all the 16 plaintext bytes and all the trials, which proves that the improved Newcache effectively defends against even the *Evict and Time* attack and the two types of *Prime and Probe* attacks, specifically crafted for it by a very knowledgeable attacker.

6. CONCLUSION

In this paper, we tested the security of a secure cache design, Newcache, using modified practical cache side channel attacks. We find that Newcache can thwart all the three representative attacks, whereas conventional set associative caches succumb easily to each of the attacks. However, we pointed out that it is important to adapt the online phase of the attacks to the cache architecture for a more thorough security evaluation, because of even smarter attackers. We redesigned attacks specifically for Newcache with the knowledge of Newcache organization. The results showed that Newcache failed to defend against the *Evict and Time* attack specialized for it, but it defeated the access-driven attacks using both type I and type II *Prime and Probe* attacks. We improved the Newcache design by moving the protection bit (the P-bit) from the tag array to the index field (the LNregs). The results showed that the improved Newcache design can completely defeat all three representative attacks, the *Evict and Time* attacks and both types of *Prime and Probe* attacks. In conclusion, this security study shows that randomization in caches works to defeat side-channel attacks, but the randomization must not be subvertible by an attacker.

Acknowledgment

This work was supported in part by DHS/AFRL contract FA8750-12-2-0295 and NSF CNS-1218817. We thank Sergey Panasyuk and Zhenghong Wang for their helpful discussions.

7. REFERENCES

[1] O. Acıiçmez and Çetin Kaya Koç. Trace-driven cache attacks on aes, 2006.

[2] D. J. Bernstein. Cache-timing attacks on aes. Technical report, 2005.

[3] J. Bonneau and I. Mironov. Cache-collision timing attacks against aes. cryptographic hardware and embedded systems. pages 201–215. Springer, 2006.

[4] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems. A practical implementation of the timing attack. In *Proceedings of the The International Conference on Smart Card Research and Applications*, pages 167–182, London, UK, 2000. Springer-Verlag.

[5] L. Domnitser, A. Jaleel, J. Loew, N. Abu-Ghazaleh, and D. Ponomarev. Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. *ACM Trans. Archit. Code Optim.*, 8(4):35:1–35:21, Jan. 2012.

[6] D. Gullasch, E. Bangerter, and S. Krenn. Cache games — bringing access-based cache attacks on aes to practice. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 490–505, Washington, DC, USA, 2011. IEEE Computer Society.

[7] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pages 388–397, London, UK, 1999. Springer-Verlag.

[8] J. Kong, O. Aciicmez, J.-P. Seifert, and H. Zhou. Hardware-software integrated approaches to defend against software cache-based side channel attacks. In *HPCA'09*, pages 393–404, 2009.

[9] R. Könighofer. A fast and cache-timing resistant implementation of the aes. In *Proceedings of the 2008 The Cryptopgraphers' Track at the RSA conference on Topics in cryptology*, pages 187–202, Berlin, Heidelberg, 2008. Springer-Verlag.

[10] R. B. Lee and Y.-Y. Chen. Processor accelerator for aes. In *Proceeding of the IEEE 8th Symposium on Application Specific Processors*, Anaheim, CA, 2010.

[11] K. Mowery, S. Keelveedhi, and H. Shacham. Are aes x86 cache timing attacks still feasible? In *Proceedings of the ACM Workshop on Cloud computing security workshop*, pages 19–24, New York, NY, USA, 2012. ACM.

[12] D. A. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: The case of aes. In *Proceedings of the Cryptographers' Track at the RSA conference on Topics in Cryptology*, pages 1–20, Berlin, Heidelberg, 2006. Springer-Verlag.

[13] D. Page. Theoretical use of cache memory as a cryptanalytic side-channel. *IACR Cryptology ePrint Archive*, 2002:169, 2002.

[14] D. Page. Partitioned cache architecture as a side-channel defence mechanism, 2005.

[15] C. Percival. Cache missing for fun and profit. In *Proc. of BSDCan*, 2005.

[16] Z. Wang and R. Lee. A novel cache architecture with enhanced performance and security. In *IEEE/ACM International Symposium on Microarchitecture*, pages 83 –93, 2008.

[17] Z. Wang and R. B. Lee. New cache designs for thwarting software cache-based side channel attacks. In *Proceedings of the annual international symposium on Computer architecture*, pages 494–505, New York, NY, USA, 2007. ACM.

[18] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 305–316, New York, NY, USA, 2012. ACM.